SIEMENS

SIMATIC NET

SOFTNET-TF/UNIX

User Manual V2.0

June 1999 **Update:** C79000-B8976-C058/01

Note

We would point out that the contents of this product documentation shall not become a part of or modify any prior or existing agreement, commitment or legal relationship. The Purchase Agreement contains the complete and exclusive obligations of Siemens. Any statement contained in this documentation do not create new warranties or restrict the existing warranty.

We would further point out, that for reasons of clarity, these operating instructions cannot deal with every possible problem arising from the use of this device. Should you require further information or if any special problems arise which are not sufficiently dealt with in the operating instructions, please contact your local Siemens representative.

General

This device is electrically operated. In operation, certain parts of this device carry a dangerously high voltage.

WARNING !

!

Failure to heed warnings may result in serious physical injury and/or material damage.

Only appropriately qualified personnel may operate this equipment or work in its vicinity. Personnel must be thoroughly familiar with all warnings and maintenance measures in accordance with these operating instructions.

Correct and safe operation of this equipment requires proper transport, storage and assembly as well as careful operator control and maintenance.

Personnel qualification requirements

Qualified personnel as referred to in the operating instructions or in the warning notes are defined as persons who are familiar with the installation, assembly, startup and operation of this product and who possess the relevant qualifications for their work, e.g.:

- Training in or authorization for connecting up, grounding or labeling circuits and devices or systems in accordance with current standards in safety technology;
- Training in or authorization for the maintenance and use of suitable safety equipment in accordance with current standards in safety technology;
- First Aid qualification.

Contents

1 INTRODUCTION TO SOFTNET-TF/UNIX 4

2 CREATING APPLICATIONS 9

2.1 Installation 9

2.2 Programming Environment 9

2.3 The SINEC TF User Interface (C Programming Interface) 14

2.3.1 Operating Modes 15

2.3.2 Supplements to the Manual SINEC TF User Interface 17

2.3.3 Porting TF-NET/UNIX Applications to SOFTNET-TF/UNIX 22

2.4 Parameter Assignment 25

3 INSTALLING AND STARTING UP APPLICATIONS 32

3.1 The Transport Name Service TNS 32

3.2 Configuration Data from stf_conf.dat 34

3.3 Configuring Application Associations 34

4 ERROR DIAGNOSTICS 40

5 SAMPLE PROGRAMS 48

6 PICS 49

7 WHO TO CONTACT 54

1 Introduction to SOFTNET-TF/UNIX

In industrial production, there are numerous hardware platforms with a wide variety of properties, for example, numeric controls, robot controls, programmable logic controllers, production controllers etc. These devices are networked with powerful network components and suitable communications hardware and software.

SOFTNET TF/UNIX provides the TF programming interface for UNIX computer systems.

☞ **You will find a detailed description of the basic ideas and concepts of the TF programming interface in the " TF Programming Interface" manual. If you are a first time user, we strongly recommend that you read this section of the documentation.**

SOFTNET Architecture

SOFTNET TF/UNIX consists of a total of four software packages:

- TFAP25 Softnet TF user interface
- APM APM interface
- CMX Communication manager UNIX,
  access to communications layer 4, addressing using logical names
- CCP-TP4 Transport provider with connectionless network

The following diagram is an overview of the product structure of SOFTNET TF/UNIX.

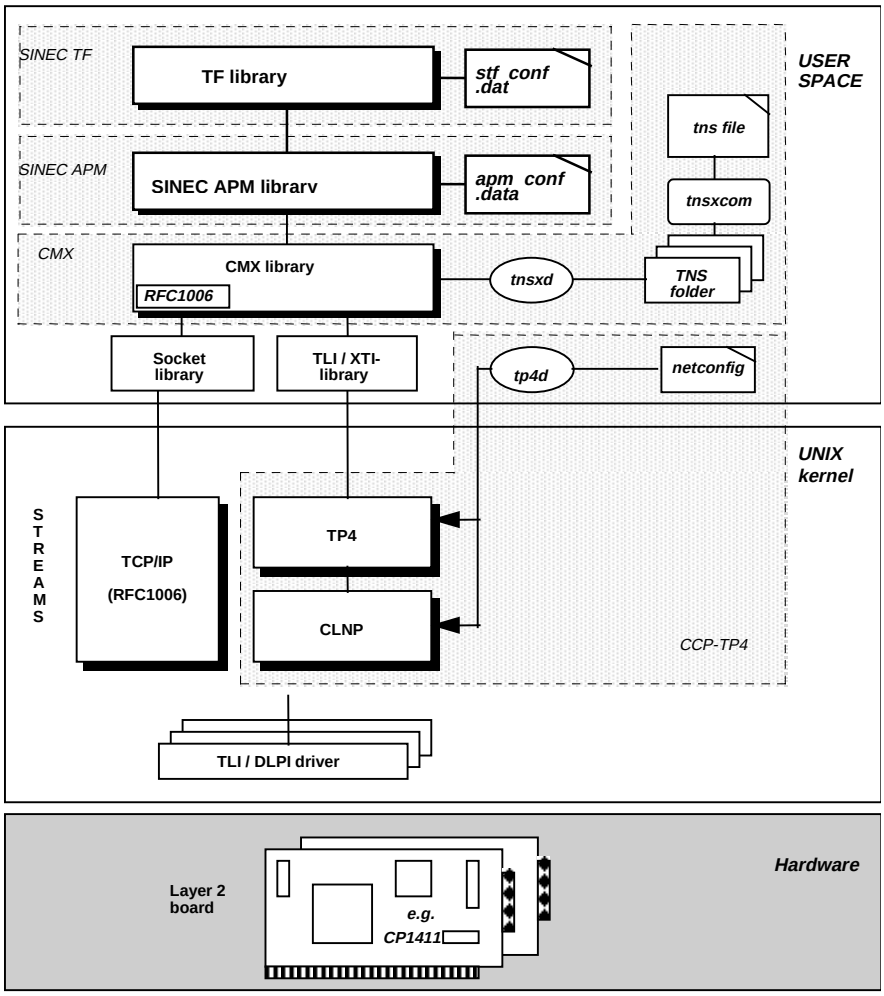

Figure 1.2: SOFTNET TF/UNIX Architecture

SINEC TF/AP

The SINEC TF/AP software package provides the SINEC TF library that is based on the SINEC APM library. The SINEC TF library contains the coding for the SINEC Technological Functions that you can use via the SINEC TF user interface, a C language programming interface.

The SINEC TF library accesses the configuration file stf_conf.dat that contains not only the general settings but also the assignment of the application association names to the connection end points. You should bear in mind that the end points are logical names in the TNS directory and that the real addresses must be assigned to the names using the tnsxcom tool (configuration).

Relevant components

Module	Function
/usr/lib/TFAP_V2.5/libstf.so	TF library
/usr/lib/TFAP_V2.5/libtfap.so	TF/AP converter library
/usr/include/TFAP_V2.5/stf_new.h	Definition of the TF interface
/usr/include/TFAP_V2.5/stf.h	Definition Part 1 of the TF user interface
/usr/include/TFAP_V2.5/stf1.h	Definition Part 2 of the TF user interface
/usr/include/TFAP_V2.5/stf_func.h	Definition of the function selection
/usr/include/TFAP_V2.5/stf_prty.h	External declaration of the TF function calls
/usr/bin/tping	Test program, to test transport connections
/usr/bin/tfping	Test program, to test application associations
/usr/stf/stf_conf.dat	TF configuration file
/usr/example/TFAP_V2.5/client.c	Example: Client application variable services
/usr//example/TFAP_V2.5/server.c	Example: Server application variable services
/usr/example/TFAP_V2.5/objtyp.h	Example: Variable definition and object description
/usr/example/TFAP_V2.5/Makefile	Example: Makefile for applications
/usr//example/TFAP_V2.5/stf_conf.dat	Example: Configuration file for TF library
/usr/example/TFAP_V2.5/apmconf.data	Example: Configuration file for APM library
/usr/example/TFAP_V2.5/tns_example	Example: TNS entry, to match stf_conf.data

SINEC APM

The SINEC APM software package provides the SINEC APM library that is based on the CMX library. SINEC APM contains the protocol handler for the SINEC AP protocol. SINEC AP is a Siemens protocol definition with optimized protocol syntax that can be compared with layers 5-7a of the ISO/OSI reference model.

The library accesses the configuration file apm_conf.dat in which various settings, such as timeout values etc. can be made.

Relevant components

Module	Function
/usr/lib/libapm.so /us/lib/libut.so /usr/lib/lbrlog.so /usr/lib/libtpdu.so /usr/bin/ro_log_on /usr/bin/ro_log_off /usr/apm/apmconf.data	APM library Utility library Logging library File access module with TPDU interface APM logging on APM logging off APM configuration file

CMX

The CMX communications manager contains the CMX library. The library provides calls for layer 4 communication in which the stations can be addressed using logical names. During operation, the logical names are assigned real addresses (T selector, MAC address) and communication takes place based on the TLI/XTI interface.

The assignment of the logical names to the real addresses is handled by the background process *tnsxd* in the TNS directory (Transport Name Service). The assignment can be made during installation and startup by the user editing an ASCII file (tns file) that is then transferred to the *tnsxcom* tool. *tnsxcom* runs a syntax and consistency check and enters the assignments in the TNS directory.

Relevant components

Module	Function
/usr/lib/libcmx.so tnsxd tnsxcom tnsxchk cmxl	CMX library TNS daemon TNS compiler Check TNS directory Trace evaluation of CMX (Activated using the environment variable CMXTRACE)

CCP-TP4

The CCP-TP4 software package provides the protocol implementations of the transport (layer 4) and network (layer 3) layers according to the 7-layer ISO/OSI reference model. Internally, it uses interfaces such as LLI/DLPI and TLI/XTI that were defined by AT&T. Some of these interfaces were included in the Xopen standard and are available for almost all UNIX systems. CCP-TP4 consists of the STREAMS multiplexers TP4 (Transport Provider Class 4) and CLNP (ConnectionLess Network Protocol) and the background process *tpd4*.

TP4 contains the protocol implementation for the transport layer, class 4. Since TP 4 is implemented as a STREAMS multiplexer it is linked into the UNIX kernel as a separate entity. As its upper access, it has the TPI (Transport Provider Interface) and can therefore be addressed via the standard interfaces available in the UNIX system, TLI (Transport Layer Interface)/XTI (Xopen Transport Interface).

TP4 uses the network provider CLNP via the NPI (Network Provider Interface). CLNP is also a STREAMS multiplexer. It contains the protocol implementation of the network layer of which SOFTNET TF/Unix uses the inactive mode.

The CLNP accesses the network via the LLI (Logical Link Provider Interface) / DLPI (Data Link Provider Interface), the layer 2 access to the network cards of the system. The Ethernet interfaces existing in the system are also addressed via the LLI/DLPI interface. Although, in contrast to TCP/IP, the CLNP uses the LLI/DLPI interface not in the Ethernet but rather in the ISO 802.3 mode, parallel operation of both protocols via the same hardware interface is possible.

The background process tp4d creates the STREAM for the communications protocol consisting of CLNP and TP4. Based on the configuration file netconfig, it also sets communications parameters that are independent of the particular connection. netconfig contains the protocol profile for SINEC that is matched to the protocol conventions of SINEC AP and must not be modified.

Relevant components

Module	Function
tp4d tp4stat	TP4 daemon Provides statistical information

2 Creating Applications

2.1 Installation

How to install SOFTNET TF/UNIX is described in detail in the product information for the operating system you use. When installing SOFTNET TF/UNIX, follow the instructions in the product information.

Parallel operation of SOFTNET-TF/UNIX and other SIMATIC NET/UNIX products is possible

Parallel operation of SOFTNET-TF/UNIX and other SIMATIC NET/UNIX products is possible. If you intend to run these packages simultaneously, only install the CMX and CCP-TP4 packages once.

2.2 Programming Environment

The "SINEC Technological Functions" (TF) are provided as C calls in the object library *libstf.so*.

The special definitions for the "SINEC Technological Functions" are located in header files that the application must include in the source file using the #*include* instruction.

The services of the "SINEC Technological Functions" can be divided into several groups. The application can deselect certain service groups using compiler options.

Header Files

The following header files are relevant and are inserted into the C source files of the application using the *#include* instruction:

stf_new.h	The header file contains the type declarations of the data types used in the TF function calls. This must be included in all the modules of an application that use the TF function calls. The status and error messages conform to the messages described in the TF manual.
stf_func.h	The content of the header file is irrelevant when using the TF user interface. Depending on the compiler options, the pointer to the TF server functions is set in this header file. This file contains C code and must be included in **one** module of the application.
stf_prty.h	This header file contains the prototypes of the TF calls. It must be included in all modules of the application that are translated with the compiler option **TF_PROTO**.

Compatibility With Old Applications

stf.h stf1.h	These two header files are intended for old applications that must be ported. These two header files were replaced by stf_new.h. The status and error messages do not conform to the messages described in the TF manual.

The header files contain the type declarations of the data types used in the TF function calls. The header file stf.h must be included in all modules of the application that use the TF function calls. The file stf.h contains the #include instruction for stf1.h

Compiler Options

The service groups can be deselected using the following compiler options:

NO_GEN	no administrative and general services
NO_SER	no serial transfer
NO_TIM	no time services
NO_VAR	no variable services
NO_DOM	no domain services
NO_PI	no program invocation services

The modules assigned to the service groups are not linked to the application.

The following switch instructs the compiler to make a prototype check of the TF calls during translation:

TF_PROTO	Check the prototypes based on the function prototypes specified in the header file *stf_prty.h*. This header file must be included in the C source files of the application using the *#include* instruction.

Linking the Applications

The TF/AP library *libstf.so* contains all the calls of the user interface as separate modules. Each application links the library modules automatically into the executable program.

Library		**Function**
dynamic *1)	static	
libstf.so	libstf.a	TF library
libtfap.so	libtfap.a	TF/AP library
libapm.so	libapm.a	APM library
libut.so	libut.a	APM utility library
librlog.so	librlog.a	APM logging library
libtpdu.so	libtpdu.a	APM file access library
libcmx.so	libcmx.a	CMX library
libsocket.so	libsocket.a	Socket library
libnsl.so	libnsl.a	Socket utility library

*1) In HP-Unix the names of the dynamic libraries have the extension "sl"

Example of an Application

The example is based on an application consisting of two modules. This application activated the prototype check. It does not support domain and PI services.

Excerpt from module 1 (prog.c):

```
# include  <stf_new.h>
# include  <stf_func.h>
# include  <stf_prty.h>

main (....
argc ..
{
    int result;

    result = tf_....(...);
```

Excerpt from module 2 (prog_1.c):

```
# include   <stf.h>
# include  <stf_prty.h>

int function_1 (....
{
    int result;

    result = tf_....(...);
```

Excerpt from the makefile:

```
##*****************************************************************************##
##                        SOFTNET H1 makefile                                  ##
##*****************************************************************************##

TF_OPT              =       -DNO_DOM -DNO_PI -DTF_PROTO
TFAP_INCL           =       /usr/include/TFAP_V2.5
TFAP_LIBS           =       -L /usr/lib/TFAP_V2.5    \
                            -lstf -ltfap             \
                            -lapm -lut -lrlog -ltpdu    \
                            -lcmx                    \
                            -lsocket -lnsl
SINEC_OPT           =       $(TF_OPT)      -I$(TFAP_INCL)

APPLICATION         =       prog.c prog1.c

prog:   $(APPLICATION) $(TFAP_LIBS)
        $(CC)   $(SINEC_OPT) -o prog  $(APPLICATION) \
        $(TFAP_LIBS)
*******************************************************************************##
```

The -L switch in the link instruction means that the linker searches for the libraries specified with the option -L in a further file directory in addition to

the standard directories. This is necessary since the TF libraries are not located in a standard directory.

2.3 The SINEC TF User Interface (C Programming Interface)

The TF user interface is defined without reference to a particular system and is described in detail in the "SINEC TF User Interface" manual. The following sections provide a general overview and contain additional information for specific systems not included in the manual.

Range of Functions

SOFTNET TF functionality is compatible with MMS, layer 7 complying with MAP 3.0 provided the application is restricted to the open services (open services = MMS compatibility). Nine service groups are defined in MMS. These services operate according to the client/server principle. The server provides a service requested by the client. When it requires a service, the client sends a request to the server. The server replies to the indication with a response. The server can also send an "unconfirmed request" to the client.

SOFTNET TF supports the following service groups:

Name in MMS standard	Name in the manual	Client	Server
Environment And General Management	Application association management	X	X
VMD Support	VMD services	X	X
Domain Management	Domain services	X	*)
Program Invocation Management	Program invocation services	X	
Variable Access	Variable services	X	X

*) Requests for load services are only supported from the network side.

A TF call exists for each service of the service groups that causes a request to be sent. The server functions are provided by the *libstf.so* library automatically. To achieve this, local TF calls exist to log on the server services and to make the server objects (for example variables) known to *libstf.so*. Further local TF calls are used for initialization, for connection and status management.

2.3.1 Operating Modes

The TF calls on the client side bring about the transfer of a request that is received as an indication by the server. After it has been processed, the server replies with a response that is received by the client as a confirmation. The parameter *mode* in the TF call decides whether the application that sent TF call waits for the confirmation.

mode = CONF_SYNC synchronous mode:
The application waits until the confirmation is received.

mode = CONF_ASYNC asynchronous mode;
The TF call is returned to the application immediately after the indication has been sent. The reception of the confirmation must be checked at a later point in time using the TF call *tf_receive()*.

Synchronous Mode

In the synchronous mode, the application waits for the confirmation from the server. During this waiting time the application is blocked and cannot handle events on the network. If you use the synchronous mode, remember the following points:

- TF calls can only be sent in the synchronous mode when no asynchronous TF calls are being processed. This means that no confirmations from the server can be received from the network.
- In the synchronous mode, the parameter *store_ind_in_sync_mode* that is set in the configuration file *stf_conf.dat* can be used to control the way in which the TF library responds to confirmed indications. It is possible to specify whether indications arriving during a synchronous TF call are acknowledged negatively or are stored temporarily for later processing. The maximum number of temporarily stored indications is determined by the parameter *max_mess_recv* that is set in the configuration file *stf_conf.dat*.
- During a synchronous TF call, unconfirmed indications are stored temporarily until the next *tf_receive()* call. The maximum number of temporarily stored indications is determined by the parameter *max_mess_recv* that is set in the configuration file *stf_conf.dat*. The waiting time until the end of the synchronous TF call is determined by the reaction time of the server.

In the synchronous mode the user can specify a maximum waiting time until the confirmation is received from the server using the parameter *ord_timeout*. **The *ord_timeout* parameter must be set higher than the timeout values of the transport system.** This is necessary so that the application receives correct error messages and can react accordingly.

TF call	ord_timeout
tf_initiate()	70 sec
tf_conclude()	70 sec
all other TF calls	70 sec

Asynchronous Mode

In the asynchronous mode, the TF call is returned immediately to the application once the indication has been sent. In the asynchronous mode, the call parameter *ord_timeout* therefore has no significance.

The application can use the waiting time until the confirmations are received from the server for other purposes. Several asynchronous jobs can be active at the same time. Per application association, however, only as many jobs are permitted as allowed by the current send credit. The application can specify the send credit for each application association itself by supplying the component *usr_snd_crd* of the structure *APPL_PATH* of the TF call *tf_get_path_ref()*.

To recognize the end of an asynchronous job, the application must call *tf_receive()*. This TF call blocks the application until the specified number of messages (*num_mess*) is received or the maximum waiting time (*wait_timeout*) expires.

The application transfers parameter fields to the TF calls using pointers. These parameter fields are accessed again in the *tf_receive()* call when the confirmation is received. This means that the parameter fields must exist during the entire processing of an asynchronous job and must not be modified. Undefined statuses result, in particular, when a parameter field was defined locally within a C procedure and the procedure is exited before the *tf_receive()* call.

2.3.2 Supplements to the Manual SINEC TF User Interface

Non-Approved Services

The TF domain and program invocation services are not approved for this version of SOFTNET TF/UNIX.

Transfer Parameters, Status Bits and Error Messages

In the TF manual, all the transfer parameters, status messages and error messages are preceded by the "TF" prefix (for example *TF_E_INIT*).

When using the header file *stf_new.h*, the transfer parameters, the status messages and error messages can be evaluated as described in the TF manual.

If you use the header file *stf.*h, the prefix "TF" must be omitted when using transfer parameters or when evaluating status and error messages (for example *E_INIT*).

New TF Call tf_getfds()

The *tf_getfds()* call returns the file descriptors occupied by the technological functions. These can be used in the *poll()* system call. This allows user programs to wait for further external events in addition to communications events. If the system call *poll()* detects events in file descriptors of TF, *tf_receive()* must then be called with *wait_timeout = 0*. Remember, that the *tf_receive()* call can return with the status message "no event occurred".

The file descriptors occupied by TF must be queried before each *poll()* call since they can change dynamically.

int tf_getfds (struct pollfds **poll_fds, int *nfds)

Parameter:

poll_fds: In *poll_fds*, the caller transfers a pointer containing a pointer to a structure *pollfds*. The function enters the currently occupied file descriptors in this structure and returns its address to the caller.

nfds: Pointer to an integer variable in which the function indicates the number of occupied file descriptors.

Return:

0: Call is OK.

The file descriptors have not changed since the last call.

1: Call is OK.
The file descriptors have changed since the last call.

-1: Error occurred.
If necessary, repeat the TF call.

If it is not necessary to wait for additional events, the return values of *tf_get_fds()* can be transferred directly to the system call *poll()*.

Example:

```
struct pollfds    *fds;
int               nfds;
int               timeout = 5000; /*5 seconds */

if (tf_getfds (&fds, &nfds) >= 0; {
      ret = poll(fds, nfds, timeout);
}
```

For a detailed description of the system call poll(), please refer to the UNIX Programmer´s Reference Manual.

Signal Processing

Internally, the TF library uses the C libraries provided by the UNIX operating system and it is not guaranteed that they can be called recursively. For this reason, when using signals, you should protect the TF calls from interruptions and must use only the UNIX system call poll() in waiting situations.

Example:

```
struct pollfds   *fds;
int              nfds;
int              timeout = 5000;
                      /* 5 sec. */
int              ret;

sighold(SIGUSR1);

result = tf_write(1, TF_CONF_ASYNC, 0,
              0, 0, TF_FALSE, opb_ptr);
sigrelse(SIGUSR1);

sighold(SIGUSR1);
ret = tf_getfds(&fds, &nfds);
sigrelse(SIGUSR1);

if(ret >= 0) {
    ret = poll(fds, nfds, timeout);
}
if (ret > 0) {
    sighold(SIGUSR1);
    ret =
     tf_receive(0,1,pb_rcv, rcv_rslt);
    sigrelse(SIGUSR1);
}
```

tf_init() and configuration file stf_conf.dat

The *tf_init()* call receives most of the settings of the configuration file *stf_conf.dat* as return parameters in the DIM_PARAM structure.
The configuration file differs in the following aspects from the description in the TF manual.

Name in stf_conf.dat	Difference compared with TF manual	Explanations
SCP_Device	Is not supported	With other products, this name specifies the transport channel of the CP card and indicates the file descriptor assigned to this channel in the parameter *scp_fd* of the structure *DIM_PARAM*.
model_name	Is not supported	Planned for future versions.
Definition of the application association	Additional function	This additional function means that the structure of the names of application associations in the application is independent of system considerations and can be freely selected.

tf_close_path()

The function of the TF call *tf_close_path()* differs from the description in the TF manual. This call causes the unconditional termination of an application association regardless of its status. After calling *tf_close_path()*, the status of the application association identified by the parameter *applref* corresponds to the status *terminated*. An application association must then be established again (depending on the application using *tf_open_path()* or *tf_initiate()*).

If the status of an application association is queried with the TF call *tf_state_path()* and if it supplies the interim statuses E4V_ESTABLISH or E4V_RECOVERY, it is possible to return to the status E4V_DOWN with the TF call *tf_close_path()* and to then re-establish the application association.

If the server side does not respond to an indication, the client side can also terminate the application association with *tf_close_path()* and then establish it again.

tf_report() tf_ustatus()

Using the TF call *tf_report()*, a server application can report the contents of one or more of its variables and with the TF call *tf_ustatus()* the logical and physical status of its virtual device to the client side without receiving an acknowledgment.

After each TF call for an unacknowledged service, it is advisable to call tf_receive(). This ensures that no problems occur with the flow control on the transport layer when there is a lot of traffic on the network.

req_msg_exch() and msg_write()

The TF call *req_msg_exch()* can be used for a bi-directional transfer of data. With the TF call *msg_write()*, an application initiates writing to a declared data area. Both TF calls can be triggered as unconfirmed services.

To ensure the transfer of the data, *tf_receive()* with *wait_timeout = 0* must be called whenever these services are called.

If this call is not made, data can be held back due to the flow control at the transport layer. The *tf_receive()* TF call detects the end of a possible bottleneck in transmission and transfers the data of the relevant service.

TF Calls tf_put_mod_ref() and tf_del_mod_ref()

Using the parameter *global_server_id=server* in the configuration file *stf_conf.dat* it is possible to log on all implemented server IDs (using *tf_put_mod_ref()*) or to log them off (with *tf_del_mod_ref()*). To do this, the name used by the parameter must be supplied to the TF function call in the object description. The name must be exactly 6 ASCII characters long.

TF call tf_get_path_ref()

Since the maximum send credit for asynchronous calls is not restricted by the TF library, the user can define it to suit the situation. For this reason, the job parameter field *APPL_PATH* of the TF function call *tf_get_path_ref()* always returns the value 0 in the parameter *usr_snd_*crd.

TF Call Parameter retry_flag

The *retry_flag* parameter decides whether a TF call is repeated automatically or is not repeated by the underlying sub-system. The repetition of a job is not permitted with the TF library, *retry_flag = TF_FALSE*.

Variable Services

A pointer to the C structure OBJ_VAR is transferred to the TF calls of the variable service either directly or indirectly via pointers to the C structures OBP_ACC and VAR_DEF. The element *v_real_ptr* of the structure OBJ_VAR is a pointer to the real data of the variable that is normally defined as a C variable. The TF calls which evaluate or modify the real data of the variable assume a certain format (byte alignment) for the C variable.

The C variable structures used for the definition of real data of TF variables must not be parenthesized with "pragma pack()".

Additional Error Messages

The following error messages are in addition to those listed in the user manual:

High Word	Low Word	Error Description
9500	0052	
A500	0052	The job is not permitted in this status. Check whether the connection still exists (using *tf_state_path()*).
9500	000B	Job number of the transmitted job is higher than the maximum permitted job number assigned in the parameters for the receiver.
9500	000C	The application association is terminated (there is no longer a connection). The connection must be re-established with *tf_initiate().*
9500	000D	The connection cannot be terminated since an asynchronous job is still active. A *tf_conclude (async.)* was triggered, but there is still an asynchronous job to be completed!
9500	0A01	Invalid connection reference
9500	0A41	Logon at the server failed.
9500	0A43	No connection established to the server.
9500	FF9D	Temporary error on the underlying APM (lack of containers). The number of available containers (parameter in *apmconf.data*) is inadequate for the initiated asynchronous jobs. Fetch the acknowledgments with *tf_receive()* and then repeat the job.

2.3.3 Porting TF-NET/UNIX Applications to SOFTNET-TF/UNIX

To ensure problem-free porting of applications that already use TF-NET/UNIX as the communication medium to SOFTNET-TF/UNIX, read the following information carefully.

TF Services

- The asynchronous TF function calls of the *local administrative services* group

  tf_aopen_path(),
  tf_aclose_path()

  are not supported by SOFTNET-TF/UNIX.

- The asynchronous TF function calls of the *VMD services group*

  tf_unso_stat()

  are also not supported by SOFTNET-TF/UNIX.

- When you change from TF-NET/UNIX to SOFTNET-TF/UNIX, you can use the synchronous TF function calls

  tf_open_path(),
  tf_close_path(),
  tf_state_path(),
  tf_ustatus()

  These functions are also executed by the library when there are still asynchronous jobs active on other application associations.

  The required response of the connection can be implemented by using the function tf_astate_path() and working with statically configured connections.

☞ **The TF domain and program invocation services are not approved for this version of SOFTNET TF/UNIX.**

Static / Dynamic Application Associations

- In contrast to TF-NET/UNIX, in SOFTNET-TF/UNIX, the transport connections for the static application association are not established when the communication module starts up, but rather connection establishment is triggered with the function tf_get_path_ref(). This means that after tf_get_path_ref(), the system waits in the tf_receive() function for connection establishment before data can be sent.

- With SOFTNET-TF/UNIX only layer 4 connections can be established statically. For reasons of compatibility to earlier versions of SOFTNET-TF/UNIX layer 7 connections must be established as dynamic application associations with tf_initiate().In dynamic application

associations, the application alone is responsible for managing the application relations (establishment, termination and maintenance).

- In TF-NET/UNIX, a send credit limits the maximum number of jobs per application association sent by a client and not yet confirmed. The application is informed of the send credit defined in the database using the TF function *tf_get_path_ref()* in the structure *APPL_PATH,* component *usr_snd_crd*.
- In SOFTNET-TF/UNIX, there is no restriction of the send credit. For this reason, the TF function *tf_get_path_ref()* in the structure *APPL_PATH,* component *usr_snd_crd* returns the value 0 to the application as the current send credit. The application can define the send credit freely.

Job Repetitions

- In TF-NET/UNIX, automatic job repetition is possible (retry_flag = TF_TRUE).
- SOFTNET-TF/UNIX does not support automatic job repetition, regardless of how the retry_flag is defined.

Asynchronous Jobs

If the asynchronous mode is used for a confirmed job, the application is continued immediately.

- In TF-NET/UNIX, a confirmation to the application indicates whether or not the message was transferred to the communications module. To receive the confirmation of the job, the tf_receive() call must be programmed in the application. If the monitoring time for the job elapses before the confirmation is received, this application is informed with a local acknowledgment.
- In SOFTNET-TF/UNIX, the application does not receive any information as to whether the message was transferred to the underlying APM protocol stack. Timeouts for asynchronous jobs are not monitored at the TF level; in other words, the *ord_timeout* parameter transferred in the function call is ignored. The application does not therefore receive any information if a job could not be completed within the time.

orderid

The relationship between an asynchronous job and a received confirmation can be recognized by the application based on the job identifier *orderid* .

- In TF-NET/UNIX, the *orderid* can use the entire range of values.
- In SOFTNET-TF/UNIX, the range is restricted since the underlying APM protocol stack uses part of it for its local management. The application must make sure that the job identifier *orderid* does not exceed a value between 0 and 65535.

Unconfirmed Jobs

- With a local confirmation to the application, the TF-NET/UNIX library confirms that an unconfirmed TF service was transferred to the communications module and that this will be handled by the module alone.
- The SOFTNET-TF/UNIX library does not return a local confirmation to the application. An unconfirmed service is accepted by the library and

passed on to the underlying APM protocol stack. To make sure that the data are sent, *tf_receive()* with *wait_timeout = 0* (polling) should be called after each call for an unconfirmed service. If this call is not made, the flow control at the transport layer may hold up data. The *tf_receive()* call detects the end of a possible bottleneck in transmission and transfers the data of the relevant service.

Querying the Time (tim_read)

With the *tim_read()* TF service, a client application can request the value of the time.

- In SOFTNET-TF/UNIX, it is not possible to request the local time. The time can only be provided by a remote server from which it is read out.

2.4 Parameter Assignment

This section describes the values that can be assigned and their effects on the TF user interface. When programming, it is advisable to take into account all the possible settings to avoid the need for program changes during installation and startup. The configuration parameters are defined in the files stf_conf.dat and apmconf.data.

Configuration File stf_conf.dat

For each SOFTNET TF application, there must be a configuration file *stf_conf.dat*. The environment variable **TF_PATH** is valid for the configuration file *stf_conf.dat*. If this is not defined, the configuration file is expected in the currently active folder.

Example:
The configuration file was renamed (*stf_conf.appl1*) and is defined in the folder */usr/config*.

TF_PATH=/usr/config/stf_conf.appl1

The configuration file *stf_conf.dat* consists of 2 parts:

- Definition of the Application Associations

  The application associations are defined in the second part of the stf_conf.dat file. The entries for this are described in section "3.3 Configuring Application Associations".

- Configuration data for the *libstf.so* library

  Using the configuration data, it is possible to adapt the *libstf.so* library for special purposes. The configuration data are read during the initialization in the TF call *tf_init()*. Based on the parameters, the *libstf.so* library requests memory dynamically.

The following parameters in the configuration file are returned to the user during the *tf_init()* function with the DIM_PARAM structure:

Name in stf_conf.dat	**Parameter in DIM_PARAM**	**Range of Values *)**	**Meaning**	**Relationship to the TF User Interface**
max_server_id	max_server_id	0 to 13 (3)	Maximum number of server IDs that can be logged on	Maximum value of parameter num_server with tf_put_mod_ref()
max_appl_rel	max_appl	1 to 255 (10)	Maximum number of application relations that can be logged on	Maximum value of parameter num_appl with tf_get_path_ref()
max_mess_recv	max_mess	1 to 32 766 (1)	Maximum number of messages that can be received per tf_receive()	Maximum value of parameter num_mess for tf_receive()
max_down_doad	max_download	0 to 32 766 (1)	With the domain services, maximum number of possible parallel download procedures for an application	
max_up_load	max_upload	0 to 32 766 (1)	With the domain services, maximum number of possible parallel upload procedures for an application	

*) The internal default is shown in brackets if the entry does not exist in the stf_conf.dat file.

The configuration file stf_conf.dat also contains parameters whose values are not indicated in the DIM_PARAM structure. These parameters are explained in the table on the following page.

Name in stf_conf.dat	Range of Values *)	Meaning	Relationship to the TF User Interface
auto_rsp_conc	TRUE	Automatic call for *tf_rsp_conclude()* with parameter conclude = TRUE in *tf_receive()*.	The application recognizes in the *tf_receive()* ***), that a CONCLUDE request was confirmed.
	<u>FALSE</u>	The application can decide whether or not to confirm or reject a CONCLUDE request.	In the *tf_receive()* ***), the application is informed that a CONCLUDE request was received that must be responded to with *tf_resp_conclude()*.
global_server_id	Exactly 6 ASCII characters **)	Logon or logoff of all the implemented server IDs depending on the setting *max_server_id*	This name must be used as the parameter *opb_ptr->server_id* with *tf_put_mod_ref()* or *tf_del_mod_ref()*.
auto_dom_serv	TRUE	Automatic call for *tf_upload()* or *tf_download()* when receiving the upload or download requests from the network.	In the *tf_receive()* ***),the file server application recognizes that the *tf_download()* or *tf_upload()*was executed internally due to a download or upload request from the network.The application must run the resulting routines (for example *tf_receive()*) just as if it had issued the TF calls itself.
	<u>FALSE</u>	The application can decide whether it calls *tf_upload()* or *tf_download()* when it receives the upload or download requests from the network.	In the *tf_receive()* ***), the application is informed that a download or upload request was received from the network and must be replied to with *tf_download()* or *tf_upload()*.
shareable	<u>TRUE</u>	DOMAIN can be used by more than one PROGRAM INVOCATION.	Parameter that is transferred to the server during startup and that can be overwritten by the TF call *tf_download()*.
	FALSE	DOMAIN can only be used by one PROGRAM INVOCATION.	Parameter that is transferred to the server during startup and that can be overwritten by the TF call *tf_download()*.

*) If the entry does not exist, the underscored default values apply.

**) If the parameter *global_server_id* does not exist, only the values for the parameters *opb_ptr->server_id* can be used for *tf_put_mod_ref()*.

****) The following table shows how the events are recognized in the TF call result = tf_receive(wait_timeout, num_mess, pb_rcv, rcv_rslt).*

Name in stf_conf.dat	Range of Values *)	Meaning	Relationship to the TF User Interface
with_va_spec	TRUE	Variable value(s) and type description are transferred to the client.	This setting is only relevant with the TF call *tf_read()*.
	FALSE	Only the variable value and not the type description is transferred to the client.	This setting is only relevant with the TF call *tf_read()*.
ignore_disc_in_ sync_mode	TRUE	A synchronous job can only be terminated by a disconnect on the local application association.	The synchronous job is terminated with result=E_CONN_LOST, in other words the local application association is closed down. The connection must be re-established and the interrupted job may have to be repeated.
	FALSE	A synchronous job on an application association can be terminated by a disconnect on any application association.	The synchronous job is terminated with result=E_CONN_LOST. At this point in time, the application has no information about which application association was interrupted. The status of the application associations must be checked, the connection re-established and if necessary the interrupted job repeated.
store_ind_in_ sync_mode	TRUE	While a synchronous job is active, indications received in the meantime are stored temporarily. The TF library does **not** acknowledge these indications negatively**.**	The application must fetch any temporarily stored jobs following each synchronous job with tf_receive. It must then evaluate the jobs and react to them accordingly. With each tf_receive(), the fetched job is acknowledged.
	FALSE	The TF library automatically acknowledges incoming jobs with a negative acknowledgment if a synchronous job is active.	

*) If the entry does not exist, the underscored default values apply.

**) If the parameter *global_server_id* does not exist, only the values for the parameters described in the manual for *opb_ptr->server_id* can be used for *tf_put_mod_ref().*

***) *The following table shows how the events are recognized in the TF call result = tf_receive(wait_timeout, num_mess, pb_rcv, rcv_rslt)*.

Event	Return Values of *tf_receive()*
If auto_rsp_conc = TRUE, a CONCLUDE request was confirmed internally.	*result = STF_OK* *pb_rcv->tf_service = CONCL* *pb_rcv->r_rslt = STF_OK*
If auto_rsp_conc = FALSE, a CONCLUDE is waiting for confirmation.	*result = E_RECVE/STP_RECV* *pb_rcv->tf_service = CONCL* *pb_rcv->r_rslt = E_SFCONCL/STP_RECV*
If auto_dom_serv = TRUE, the TF call tf_download() was executed internally after a download request from the network.	*result = STF_OK* *pb_rcv->tf_service = REQ_DLOAD* *pb_rcv->r_rslt = STF_OK*
If auto_dom_serv = TRUE, the TF call tf_upload() was executed internally after an upload request from the network.	*result = STF_OK* *pb_rcv->tf_service = REQ_UPLOAD* *pb_rcv->r_rslt = STF_OK*
If auto_dom_serv = FALSE, a download request from the network is waiting to be processed.	*result = E_RECVE/STP_RECV* *pb_rcv->tf_service = REQ_DLOAD* *pb_rcv->r_rslt = E_SFLOAD_REQ/STP_RECV*
If auto_dom_serv = FALSE, an upload request from the network is waiting to be processed.	*result = E_RECVE/STP_RECV* *pb_rcv->tf_service = REQ_UPLOAD* *pb_rcv->r_rslt = E_SFUPLOAD_REQ/STP_RECV*

The response of the static connections is controlled using the following parameters in the configuration file:

Name in stf_conf.dat	**Range of Values**	**Default**	**Meaning**
t_retry	0 .. 65535	100 s	Wait time in seconds from the beginning of a failed connection establishment to the next attempt.
t_recovery	0 .. 65535	10 s	Wait time in seconds after closedown/abort until the next connection establishment attempt.
retry_count	0 .. 4.3 *10^9	500000	Maximum number of connection establishment retries following an abort/close.

Example: Content of an stf_conf.dat File

```
auto_rsp_conc = TRUE      /* default FALSE */
global_server_id = xxxxxx /* default no default */
auto_dom_serv = TRUE      /* default FALSE*/
shareable = FALSE         /* default TRUE*/
max_server_id = n         /* default 3*/
max_appl_rel = n          /* default 10*/
max_mess_recv = n         /* default 1*/
max_down_load = n         /* default 1*/
max_up_load = n           /* default 1*/
```

Configuration File apmconf.data

For each SOFTNET TF application, there must be a configuration file *apmconf.dat*a for the separate package SINEC APM.
The environment variable **APM_PATH** is valid for the configuration file *apmconf.data*. If this is not defined, the configuration file is expected in the currently active folder.

Example:
The configuration file was renamed (*apm_conf.appl1*) and is defined in the folder */usr/config*.

APM_PATH=/usr/config/apm_conf.appl1

Using the configuration file it is possible to adapt the APM library for user-specific purposes. The parameters are read during the initialization in the APM function call *ap_init()* and the library is adapted dynamically.

Some of the parameters of the configuration file must be matched to the parameters of the TF configuration file *stf_conf.dat*. The configuration file contains the following parameters:

Name in apmconf.data	Range of Values *)	Meaning	Relationship to the TF User Interface
assoc_limit	1 - 255 (1)	Number of permitted applications per process	must be set to 1 for SOFTNET TF applications.
appl_limit	1 - 255 (16)	Number of connections per process	Must be matched to the definition in the configuration file *stf_conf.dat*, parameter *max_appl_ref*.
time_limit	0 - 32767 (60)	Monitoring time of an asynchronous job	Monitoring time of a TF call. Acknowledgments arriving later are automatically discarded.
try_limit	0 - 32767 (0)	Repetitions per job; 0 means no repetition	SOFTNET TF does not support automatic repetitions of a TF call.
open_interface	0 - 1 (1)	Definition of the interface 0 = IROS interface 1 = open interface	SOFTNET TF requires the open interface.
pdu_length	512 -10248 (4096)	Length of the APM-PDU	SOFTNET TF works with a default PDU length of 4096 Bytes.
num_cont	20 - 32767 (200)	Number of available containers	The number must be matched to the maximum possible number of pending asynchronous TF jobs.

*) The default values are indicated in brackets.

3 Installing and Starting Up Applications

General

SOFTNET TF applications use the C interface to the SINEC Technological Functions (TF user interface) for communication with remote systems (for example programmable controllers). To achieve this, the application must link the libraries *libstf.so* and *libtfap.so*. The response of certain functions of the *libstf.so* library and their response on the TF user interface can be controlled using parameters of the configuration file*stf_conf.dat* (see section "2.4 Parameter Assignment").

On the TF user interface, the application uses freely selectable names for the application associations to address partner applications. The assignment of the actual address parameters of the partner applications to the application associations is then made while the program is running.

The application association names are defined in the configuration file *stf_conf.dat*. The definitions referred to entries of TNS (Transport Name Service). In the TNS, the logical names are assigned the communication addresses (local and remote t-selector, remote Ethernet address etc.).

3.1 The Transport Name Service TNS

Transport Name Service (TNS)

Within the transport name service, SOFTNET TF manages the address information of the communications stations. This information must be consistent within a system. It is managed by the background process *tnsxd* as TNS entries in the TNS directories. Each TNS entry is assigned a logical TNS name that must be unique within the computer system.

Creating the TNS Entries

The TNS entries are created using the TNS compiler *tnsxcom*. The address information is edited in an ASCII file that is transferred to the TNS compiler as a parameter.

- Call the TNS compiler to create TNS entries:

  tnsxcom -u ASCIIfile

  (This call must be made as a superuser.)

- Check the newly created entries:

  tnsxcom -D ASCIIfile

  (This call does not need to be made as a superuser.)

Modifying Existing TNS Entries

If you need to change the address information of existing TNS entries, follow the procedure below:

- Read the current TNS entries into an ASCII file:

```
tnsxcom -D ASCIIfile
```

(This call does not need to be made as a superuser.)

- Modify the address information in the ASCII file:

  Refer to the structure of TNS entries

- Update the TNS entries by calling the TNS compiler:

```
tnsxcom -u ASCIIfile
```

(This call must be made as a superuser.)

- Check the modifications made:

```
tnsxcom -D ASCIIfile
```

(This call does not need to be made as a superuser.)

Deleting Existing TNS Entries

You delete an existing TNS entry as follows:

- Generate an ASCII file with the following entry:

```
TNSname DEL
```

(Where TNSname = the TNS entry to be deleted)

- Call the TNS compiler:

```
tnsxcom -u ASCIIfile
```

(This call must be made as a superuser.)

- Check that entries have been deleted:

```
tnsxcom -D ASCIIfile
```

(This call does not need to be made as a superuser.)

Outputting the Address Information of All TNS Entries

The current address information of all existing TNS entries can be written to an ASCII file using the following command:

```
tnsxprop > ASCIIfile
```

(Where ASCIIfile = any file name)

3.2 Configuration Data from stf_conf.dat

For each SOFTNET TF/UNIX application, there must be a configuration file *stf_conf.dat*. Apart from the configuration parameters (see section 2.4 Parameter Assignment")this also contains the definition of the application associations.

3.3 Configuring Application Associations

An application association is the communications path between two applications located on remote systems. The applications reference the application associations using names in the TF call *tf_get_path_ref()* on the TF user interface. This section describes the assignment of address information of an application association to the name of the application association within an application.

The assignment takes two steps:

- In the configuration file *stf_conf.dat*, each referenced application association name of the application is assigned logical names of the Transport Names Services (TNS).
- The address information of applications between which there is an application association is stored using the logical TNS names. The address information is stored with the *tnsxcom* tool. As its input, the tool uses a readable file in which the TNS entries can be edited.

Application Association Names in stf_conf.dat

The application names must be defined in the *stf_conf.dat* . These are the same names that the application transfers in the TF call *tf_get_path_ref()*. The application association names can be defined using 2 methods:

1. Definition as AP connection using key words

   This allows the application association to be defined as static and also allows it to be distinguished from a redundant application association.

2. By assigning the connection end points

   The application association names are assigned TNS entries as local and remote connection end points using logical TNS names.. Due to the conversion in the configuration file *stf_conf.dat*, the application association names can be selected freely regardless of the logical TNS names of other applications.

   ☞ **This method supports only dynamic connections at layer 7 (= connections, established with INITIATE).. These are provided only to assure compatibility with previous versions and should not be used any longer with new applications.**

Application Association Name Format in stf_conf.dat

☞ **Application association names can contain blanks. For this reason, the colon following the application association name is mandatory since this indicates the end of the name. If more than one blank is necessary at the end of an application association name, these must be inserted between the last character of the connection name and the colon.**

AP Connections Using Keywords

An application association is defined using the following keywords:

```
Type            = AP                    # Type of connection
                                        # is always AP
appl_assoc_name =                       # Application association name
                <Appl.assocname>:       # = logical TNS name
appl_type       = <static | dynamic>    # Type of application connection
                                        # Possible values:  static
                                        #                   dynamic
con_mode        = <active | passive>    #
                                        # Possible values:  active
                                        #                   passive
```

<Appl.BezName> Name of the application association (*tf_get_path_ref*()) is also the logical TNS name

The corresponding TNS entry must contain a TSEL and a TA element.

<static | dynamic> Type of application association

static: If the application association is active, establishment is triggered automatically in *tf_get_path_ref()* (only layer 4 connections) USER1PR = 00 must be set in the TNS entry

dynamic: With active application associations, the establishment must be triggered by the user with *tf_initiate()* or *tf_open_path()*. USER1PR = 01 must be set in the TNS entry

<active | passive> Initiative When Initializing an Application Association

active: The local application has the initiative for connection establishment USER2PR = 01 must be set in the TNS entry

active: The local application waits for connection establishment USER2PR = 00 must be set in the TNS entry

Application Association Names by Defining End Points

Per application association, there is a separate TNS entry for the local and for the remote end point or there is one entry for both end points. Using the parameters of the local end point, the application logs on at its local communications system with *tf_get_path_ref()*. With the parameters of the remote end points, either a *tf_initiate()* is executed or the arrival of an INITIATE request is expected from the partner in *tf_receive()*.

Definition using end points as the following format:

<Appl.assocname>: <local_endpoint> <remote_endpoint>

<Appl.assocname>	Name of the application association (*tf_get_path_ref*())
<lokalerEndpunkt>	Name for the TNS entry of the local end point The TNS entry must be a TSEL element and the elements USER1PR and USER2PR
<remote_endpoint>	Name for the TNS entry for the remote end point The TNS entry must contain a TA element and the USER1PR and USER2PR elements.

Structure of the TNS Entries in the ASCII File

The TNS entries contain the following address information:

- Logon of the local application association (TSEL element)
- Transport address of the remote application (TA element)
- Initiative for establishing application association (USER2PR element)
- Type of application association (USER1PR element)

```
TNSnamel            # logical TNS name
                    # Name must match the entry in stf_conf.dat
                    # maximum of 30 ASCII characters.

TSEL lantype tsel   # local logon of the application association
                    # Network layer (lantype) and local
                    # Transport selector (tsel); mandatory
                    # Parameter; must be set
                    # Possible values lantype:
                    #   LANSBKA      Null Internet protocol
                    # Possible values for tsel:
                    #   A'tsap' tsap =  max.8 characters (ASCII coded)
                    #   X'tsap' tsap =   max. 8 octets  (hex coded)

TA subnet_id lantype adr tsel
                    # Transport address of the remote application
                    # local Ethernet card (subnet_id),
                    # Network layer (lantype),
                    # Ethernet address of the partner system (addr)
                    # and transport selector (tsel) of the partner
                    # application;
                    # Possible values lantype:
                    #   LANSBKA      Null Internet protocol
                    # Possible values adr:
                    #    Ethernet address (hex)  exactly 6 octets
                    # Possible values for tsel:
                    #   A'tsap' tsap =  max.8 characters (ASCII coded)
                    #   X'tsap' tsap =   max. 8 octets  (hex coded)

USER1PR\            # Type of application association
    nn              #Possible values nn:
                    #    00  static connection
                    #    01   dynamic connection
                    #         Triggered by the application

USER2PR\            # Initiative When initializing an application
    nn              relation
                    #Possible values nn:
                    #    01   active) initiative by loc. application
                    #    00    (passive) initiative by remote application

USER3PR\            # Content is not relevant for SOFTNET TF
                    # must, however, have 00 entered
                    # to ensure consistency
```

subnet_id

When using the OSI transport protocol, SOFTNET TF/UNIX supports operation with more than one Ethernet board. This parameter is used to address the Ethernet board used in active connection establishment to the remote SOFTNET TF/UNIX application.

☞ Caution: The <subnet_id> parameter depends on the particular operating system. Please refer to the SIMATIC NET/UNIX product information for the operating system you are using to check the correct value.

The following format is mandatory for <subnet_id>:

X'<string>' <string> character string with exactly 4 hexadecimal digits

Restrictions Affecting Names in the TNS Entries

- The **logical name of a TNS entry** can be a maximum of 30 characters long. It must start with a letter or the underscore character, the remaining 29 characters can be letters, underscores or numbers.
- The **transport selectors** can be either in ASCII format when preceded by an "A" or in hexadecimal format when preceded by an "X".
- When using the **ASCII format**, a maximum of 8 characters are possible for the **transport selector**. Permitted characters are letters, numbers or the underscore character.
- When using the **hexadecimal format** for the **transport selector**, an even number of a maximum 16 hexadecimal digits can be used. Hexadecimal numbers are made up of the numbers 0 to 9 and the letter a to f or A to F.

Configuration Samples **Configuring using Keywords**
The static, active application relation OS00001 is configured.

Content of stf_conf.dat

```
/*****************************************************************/
/* Response of static connections                                */
/* (valid for all application relations                           */
/*****************************************************************/

t_retry     = 100     /* 100 s wait time between establishment
attempts*/
t_recovery  = 10      /* 10 s wait time after connection abort    */
retry_count = 500000  /* Give up after 5000000 attempts           /*

/*****************************************************************/
/* Definition 1st application relation                            */
/*****************************************************************/
Type              = AP*/
appl_assoc_name  = OS00001:
appl_type        = static
con_mode         = active

/*****************************************************************/
/* Further connections                                           */
/*****************************************************************/
```

Content of TNS:

```
OS00001\
        TA      LANSBKA 080006011111 X’0301030201202020’
        TSEL    LANSBKA X’0102030103202020’
        USER1PR\
                00
        USER2PR\
                01
        USER3PR\
                00
```

Definition via endpoints
The active application relation A_Active is configured.

Content of stf_conf.dat

```
A_Active:  loc_TNS_Name    rem_TNS_Name
```

Content of TNS: local endpoint

```
loc_TNS_Name\
        TSEL    LANSBKA X’0102030103202020’
        USER1PR\
                00
        USER2PR\
                01
        USER3PR\
                00
```

Content of TNS: remote endpoint

```
rem_TNS_Name\
        TA      LANSBKA 080006011111 X’0301030201202020’
        USER1PR\
                00
        USER2PR\
                01
        USER3PR\
                00
```

4 Error Diagnostics

The following sections describe the tools and logging mechanisms for diagnosing error groups. The errors that can occur when using SOFTNET TF applications can be divided into two groups, as follows:

Communication Errors

The main causes and effects of communication problems are as follows:

- Application associations cannot be established. The most common causes are problems with the underlying transport system (for example transport name service not started).
- Individual application associations cannot be established. The cause is usually in the configuration or the limit values of the transport system (for example maximum number of connections) have been exceeded.
- Application associations break down during operation. These problems can be caused by longer network failures, incorrect response of the partner or by the application not calling *tf_receive()* often enough.
- Problems during the data exchange between applications. The cause is usually an incorrect response on the partner or that *tf_receive()* is not called often enough.

Programming Errors

Common programming errors are as follows:

- Incorrect parameters are transferred to the TF functions (for example the parameter *appl_ref* to a non-existent application association).
- The values for the parameter *ord_timeout* have been selected too low in the synchronous mode.
- In the asynchronous mode, the object descriptions are changed before the service is completed.
- Rare external events (errors, exceptional situations) have not been taken into account. For example, server services may be logged on that cannot be handled.
- The TF function *tf_receive()* is not called often enough.

Troubleshooting in Communication

CCP-TP4 Transport System

The STREAM that forms the protocol stack is maintained by the *tp4d* background process.

Using the UNIX command *ps*, you can check whether the tp4d process is active. The *tp4stat* command is also available with which you can fetch and analyze statistical information from the transport system.

Transport Name Service (TNS)

In the TNS, the background process *tnsxd* manages the address information of the applications involved in communication as TNS entries in the TNS directory.

With the UNIX command *ps* you can check whether the *tnsxd* process is active.

The existing TNS entries of the TNS directory can be read into the ASCII file with the name ASCIIfile using the command

```
tnsxcom -D ASCIIfile
```

and checked with the command

```
tnsxcom -s ASCIIfile
```

CMX Library Trace

The trace of the CMX library is controlled by the environment variable *CMXTRACE*. By supplying the environment variable with a value, the trace is activated and the scope of the information to be collected is specified.

The trace entries of a process are collected as compact binary data in a dynamically created buffer and periodically written to temporary files. These binary files are edited separately with the *cmxl* **tool.** The binary files are saved in the directory */usr/tmp*. The file names consist of the prefix *CMXLa* or*. CMXMa* and the process identification number *pid*.

cmxl reads the entries generated by the trace from the temporary file. The scope of the analysis is decided by the options selected for *cmxl*.

CMXTRACE: Controlling the Trace

The options specified in CMXTRACE control the trace. The options *s*, *S*, and *D* determine what is logged. The options *p*, *r* control the buffering and (wrap) writing of the file:

```
CMXTRACE = "[ -s] [-S] [ -D] [ -p fac] [ -r wrap] [ -f directory]"
```

-s The CMX calls, their arguments, the options and user data are logged normally.

-S The calls, their arguments, the contents of any options, the user data in their full length are logged.

The options -s and -S exclude each other.

-D The calls with additional information about system calls are logged in detail. This option is only available in addition to -s or -S.

-p *fac* The decimal digit *fac* determines the buffer factor. The amount of buffering is determined according to *fac* * BUFSIZ where BUFSIZ is determined by <stdio.h>.

If you specify *fac* = 0, each trace entry is written to the file immediately (with no buffering).

fac = 0..8

Default: *fac* = 1

-r *wrap* The decimal number *wrap* specifies that after

wrap * BUFSIZ bytes (BUFSIZ according to <stdio.h>)

logging continues in the second temporary file <directory>/CMXMa<pid>. This second file handles the trace in exactly the same way as CMXLa<pid>. After *wrap* * BUFSIZ bytes, the trace switches between the two files. Following this switch over the content of the file is overwritten.

Default: *wrap* = 128

-f *directory* The trace files are written to the specified directory.

Default: *directory*: /usr/tmp or /var/tmp

cmxl: How the Trace Trace

cmxl reads the entries generated by the trace from the temporary *file*, processes the entries according to the selected options and outputs the result to *stdout*.

The following options specify which trace entries from the *file* are processed. It is possible to specify more than one of the values described below per *cmxl* call. Only the options *v* and *x* exclude each other. If no options are specified, *cdex* is used as the default.

cmxl [-c] [-d] [-e] [-t] [-x] [-D] file

-c The CMX calls for logging on and off the TS application with CMX and for establishing and terminating the connection are processed.

-d The CMX calls for data exchange and flow control are processed.

-e The CMX calls for handling events are processed.

-t	In addition to logging the error messages, the t_error() calls are processed explicitly. Error messages are always logged even if this option is not specified.
-v	The CMX calls, their arguments, the options and the user data are processed in detail. The extent of processing depends on the options specified for CMXTRACE.
-x	The calls and their arguments are processed without options and user data.
-D	This option selects detailed processing with additional information about system calls.
file	Name of one or more files containing trace entries to be processed.

Example of Activating and Evaluating the CMX Library Trace

Example of a configuration for activating the CMX library trace:

```
csh:   setenv CMXTRACE "-SD -p 0 -r 64 -f ."
sh:    CMXTRACE="-SD -p 0 -r 64 -f ."; export CMXTRACE
```

Example of a configuration for editing the trace files:

```
cmxl   -Dv CMXLa<pid> > file_name
```

It is advisable to redirect the data to a file, otherwise they are output to *stdout*.

☞ **CMX error codes and a brief description can be found in the CMX header file */usr/include/cmx.h*.**

SINEC APM Logging

The SINEC APM protocol underlying SINEC TF includes a dynamic logging system. The logging system consists of basic logging and a section that allows dynamic change of the logging levels at any time. The user has the choice between the highest logging level (log all = level 1) or no logging (do not log = level 4). A file must contain the required logging level.

To support the dynamic logging system, the user must define an environment variable containing the path and file name of a global logging level file.

The process-specific logging level file is used to change the level for a particular process during operation. For this, there are commands available that generate the global logging level file when the first call is made.

With the example configurations, the library trace can be switched on and data can then be analyzed.

For more detailed information, refer to the SINEC APM user manual (see References).

The global logging level file *roslog* that contains the default logging level is required in the */tmp* directory. This file is generated when the start or stop command is first called.

ROS_LLFILE = /tmp/roslog

The logging system is activated by the following call:

ro_log_on <parameter>

Parameter:
1 level high

Everything is logged (function calls of the user interface and internal calls of the SINEC-APM)

2 level medium

Logging of inputs into and outputs out of user interfaces - function calls

3 level low

Only errors are logged

By activating the logging, two logging files are created and written to alternately. These logging files are created in the */tmp* directory under the names *humba.L1.<pid>* and *humba.L2.<pid>* (*pid =* process identification). Both files are ASCII files and can be analyzed directly using an editor.

Enter the following to terminate the logging function:

ro_log_off

This command does not require a value for the logging level. This command automatically writes logging level 4 (no logging) into the logging level file.

Testing the Transport Connection (tping)

Communication at the transport layer can be checked with the *tping* program. The program uses the transport name service.

tping [-o tnsname1] [-p tnsname2]

Parameter:
-o tnsname global name for the TNS entry for the logon at the local transport system; if the parameter is not specified, the default *tping* applies.

-p tnsname2 TNS entry with which it is attempted to establish a transport connection; if the parameter is not specified, *tnsname1* is used.

The program logs on at the local transport system with the TNS entry *tnsname1* and attempts to establish a transport connection with the TNS entry *tnsname2*. Three reactions to the connection request are possible:

- **Connection request accepted**
  Message from tfping:

  ```
  "T_CONF received after <time> seconds.
   Connection to remote system established!!!"
  ```

  The connection request was confirmed with Connect Confirm by the partner. This proves that the transport system is functioning and that the parameters of the transport layer are correctly set. *tping* terminates the connection again with DISCONNECT and logs off at the transport system.

- **Connection request rejected**
  Message from tfping:

  ```
  "T_DISIN received after <time> seconds... returned code:
   <error number>: <error description>"
  ```

  The connection request was rejected with Disconnect by the partner. Communication via the transport system is functioning. If a transport connection to this partner is required, parameters must be adapted to the transport layer.

- **No reaction**
  Message from tfping:

  ```
  "T_DISIN received after <timeout> seconds... returned code:
   <error no.>: Connection cannot be set up because partner does not
                respond to CONRQ"
  ```

  Possible causes include:

  - There is not partner in the network with this network address or the partner is not operational.

  - The partner is configured so that it does not react to incorrect transport layer parameters (for example TSAP).

The functionality of the transport system can be checked using a LAN analyzer. The transport system is functioning when the LAN analyzer records the appropriate connect request PDU.

☞ **The CMX error codes (*reason*) and a short description can be found in the CMX header file */usr/include/cmx.h*.**

☞ **Errors can be investigated in greater detail with the CMX trace tool.**

Testing the Application Associations (tfping)

Using the *tfping* program, it is possible to check whether application associations are correctly configured and whether they can be established to a partner. The program uses the configuration file *stf_conf.dat* in the local directory to decode the application association name. It provides the following three modes:

tfping [-v]

tfping -d [filename] [-v]

tfping TNS-name1[,TNS-name2,TNS-name3,...] [-v]

Parameter:	***-v***	Display extended monitor output.
	without	Interactive mode
	TNS names	Selective test (batch)
	-d [filename]	Complete test.

- **Interactive**
  The program is called without parameters. It prompts the name of the application association to be tested. Several names must be separated by commas.
- **Batch**
  The names of the application associations are supplied to the program as parameters when it is called. The names must also be separated by commas here.

  Example: `tfping Aname_1,Aname_2,Aname_n`
- **Complete test**
  The program is supplied with a configuration file as a parameter. It checks all the application association names of the configuration file. If no file name is specified, the file specified by the environment variable TF_PATH is processed. Normally, this is the file *stf_conf.dat*.

  Example: `tfping –d stf_conf.dat`

Functions:

Using the application association names, the program attempts to establish an application association to the corresponding partners using *tf_initiate()*.

If an application association is defined as active in the TNS entry, *tfping* establishes the association with *tf_initiate()*, reads the identity of the partner (vendor, device type and revision) using *tf_identify()* and then terminates the application association again.

If *tfping* recognizes a passive application association in the *tf_initiate()*, it executes the server functions of the TF services *tf_initiate(), tf_identify()* and *tf_conclude()*. In the *tf_receive()*, the program waits for the partner to establish the application association, if appropriate to read out the identity and then to terminate the association again.

The *tfping* program terminates itself automatically as soon as all the application associations have been closed.

☞ **The program can be aborted with the DEL key (for example when the partner does not react).**

Possible errors when logging on an application association:

The application association name is not entered in the *stf_conf.dat* configuration file *---> tf_get_path_ref()* indicates the result *Fatal_Error* and *tfping* is terminated.

Possible errors when initiating an application association:

The partner does not exist; for example a partner application has not been started, the cable is not plugged in, the CPU + CP of the PLC are in the STOP mode *---> tf_initiate()* indicates the result *Abort* and *tfping* is terminated.

Note: If the remote system is configured as being active and only establishes a level 4 connection once, no message is output.

5 Sample Programs

The programming examples are in the directory */usr/example/TFAP_V2.5*. With the example programs and the supplied configuration files, you can run simple client/server communication locally on your computer.

The example programs consist of the following files:

client.c	**Client module**
server.c	**Server module**
objtyp.c	**Object description of the variables**

With the supplied *makefile*, you can translate the programs and link them to the libraries. This produces the runnable programs *client* and *server*.

To initialize the communication system, the directory also contains the two configuration files

stf_conf.dat

apmconf.data.

These configuration files have been adapted to the two example programs and do not need to be modified.

The *tns_example* file contains an example TNS entry. This must be adapted to the partner system. With the TNS command *tnsxcom -u tns_example*, you transfer the TNS entry to the TS directory.

6 PICS

The range of services of the TF **Library** is described in the form of PICS (Protocol Information Conformance Statements). The following conventions are used:

X	Means that a client and server module is implemented.
C	Means only a client module is implemented.
S	Means only a server module is implemented.
-	Means that the function is not approved in this version.
*	Means further details in a PICS must be taken into account
**	No segmentation

PICS 1 (General)

PICS 1	
Vendor name	SIEMENS AG
Model name	SOFTNET TF/UNIX
Revision	V2.5A09
Device/version number	
Operating system	Solaris V2.4 - V2.6 SCO-UNIX V5.0 - V5.0.4 HP-UX V10.10 - V10.20
Abstract syntax Transfer syntax (PROTID)	
Version number	
Calling user (yes or no) Called user (yes or no)	Yes Yes
List of standardized names	
Companion standards - Abstract syntax - Version number	None

PICS 2
TF Services

PICS 2	
Variable services	
Read	X
Write	X
Report	S
Get variable attributes	X
Domain management	
Initiate up/download	-
Up/download segment	-
Terminate up/download	-
Request upload sequence	-
Upload segment	-
Request upload sequence response	-
Request download sequence	-
Request upload sequence	-
Load domain content	-
Store domain content	-
Delete domain content	-
Get domain attributes	-
Program invocation management	
Create PI	-
Delete PI	-
Start PI	-
Stop PI	-
Resume PI	-
Reset PI	-
Get PI attributes	-
VMD services	
Status	X
Unsolicited Status	X
Get name list	X
Identify	X
Get capability list	X
Application association management	
Initiate	X
Conclude	X
Abort	
Time functions	
Request time	C
Serial transfer	
Read byte string	X
Write byte string	X
Transparent data exchange	X

PICS 3 (Data Types)

PICS 3		
Basic data types	a) boolean b) bitstring c) integer d) unsigned e) octet string f) visible string g) floating point	X X X X X X X
	arrays	X
	structures	X
Named variable	all scopes defined by TF	
Unnamed variable		
Nesting level Hierarchiestufen (=nesting)		2
Number of alternative accesses (number of alternative accesses in a job)		16
List of variables (number of variables in a job)		16
Relationship:	Object description to access description in job	The object descriptions in the server have the maximum complexity of access definitions in the protocol (see above).

PICS 4 (Value Ranges)

PICS 4	
Range for floating point values	s. See appendix A in the TF User Interface manual
Range for floating point exponent	s. See appendix A in the TF User Interface manual
Range for floating point	s. See appendix A in the TF User Interface manual
Range for integers	s. See appendix A in the TF User Interface manual
Range for unsigned	s. See appendix A in the TF User Interface manual
Maximum length BIT_STRING	0 <= data_resv <= 2**31-1
Maximum length OCTET_STRING	0 <= data_resv <= 2**31-1 can be negative
Maximum length VISIBLE_STRING	0 <= data_resv <= 2**31-1 can be negative
TF address	
Max. job management time	2**31-1 milliseconds
Capability list of the virtual device	Not used

7 Who to Contact

Contacts for Technical Questions

If you have questions about using this product, please call our SIMATIC NET hotline in Nuremberg:

Siemens AG
Customer Support

Telephone: ++49-911-895-7000
Telefax: ++49-911-895-7001

The following list shows you who to contact in your area:

Germany	Aachen Postfach 12 85 52013 Aachen Hr. Görgens Tel. (0241) 451 - 252 Fax (0241) 451 - 398	Braunschweig Ackerstraße 20 Postfach 33 47 38023 Braunschweig AUT P12 Hr. Reupke Tel. (0531) 7012 - 436 Fax (0531) 7012 - 400
	Augsburg Werner-von-Siemens-Str. 6 Postfach 10 23 49 86135 Augsburg AUT Hr. Gleichfeld Tel. (0821) 2595 - 371 Fax (0821) 2595 - 412	Bremen Contrescarpe 72 Postfach 10 78 27 28078 Bremen AUT P12 Hr. Kroll Tel. (0421) 364 - 2431 Fax (0421) 364 - 2842
	Bayreuth Postfach 10 10 51 Weiherstr. 25 95410 Bayreuth AUT 51 Hr. Hüttl Tel. (0921) 281 - 246 Fax (0921) 281 - 444	Düsseldorf Lahnweg 10 Postfach 11 15 40002 Düsseldorf AUT FG15 Hr. Kreienmeier Tel. (0211) 399 - 1412 Fax (0211) 399 - 1848
	Berlin Schwarzer Weg 3 14532 Kleinmachnow (Berlin) AUT P11 Hr. Schulze Tel. (030) 3993 - 3001 Fax (030) 3993 - 2582	Essen Kruppstraße 16 Postfach 10 33 63 45128 Essen AUT P14 Hr. Ender Tel. (0201) 816 - 2925 Fax (0201) 816 - 2344
	Bielefeld Schweriner Straße 1 Postfach 78 20 33605 Bielefeld AUT Hr. Klein Tel. (0521) 291 - 518 Fax (0521) 291 - 506	Frankfurt a. Main Rödelh. Landstr. 5-9 Postfach 11 17 33 60 052 Frankfurt AUT VG 21P Hr. Wasel Tel. (069) 797 - 3825 Fax (069) 797 - 3442

	Hamburg Lindenplatz 2 Postfach 10 56 09 20038 Hamburg AUT P11 Hr. Becker-Ullmann Tel. (040) 2889 - 2931 Fax (040) 2889 - 3209	Regensburg Hornstraße 10 Postfach 10 09 45 93009 Regensburg AUT P/S12 Hr. Bauer Tel. (0941) 4602 - 226 Fax (0941) 4602 - 236
	Hannover Hildesheimer Straße 7 Postfach 53 29 30876 Laatzen AUT S22 Hr. Weidmann Tel. (0511) 877 - 2448 Fax (0511) 877 - 2113	Saarbrücken Martin-Luther-Straße 25 Postfach 10 28 42 66123 Saarbrücken AUT VG P Hr. Laufer Tel. (0681) 386 -2479 Fax (0681) 386 - 2111
	Köln Franz-Geuer-Straße 10 Postfach 30 11 66 50781 Köln AUT FG10 Hr. Boxberg Tel. (0221) 576 - 3724 Fax (0221) 576 - 2795	Stuttgart Weissacher Straße 11 Postfach 10 60 26 70499 Stuttgart AUT A12 Hr. Schrickel Tel. (0711) 137 - 2028 Fax (0711) 137 - 2684
	Chemnitz Bornaer Str. 205 Postfach 400 546 3 09114 Chemnitz AUT P21 Hr. Mehner Tel. (0371) 474 - 3512 Fax (0341) 210 - 3525	Würzburg Andreas-Grieser-Straße 30 Postfach 32 80 97042 Würzburg AUT Hr. Tasch Tel. (0931) 6101 - 376 Fax (0931) 6101 - 542
	Mannheim Dynamostraße 4 Postfach 20 24 68028 Mannheim AUT Hr. Kopplin Tel. (0621) 456 - 2851 Fax (0621) 456 - 2545	München Richard-Strauß-Straße 76 80286 München AUT P13 Hr. Wildung Tel. (089) 9221 - 4060 Fax (089) 9221 - 4399
	Nürnberg Von-der-Tann-Straße 30 90327 Nürnberg AUT A13 Hr. Glas Tel. (0911) 654 - 3587 Fax (0911) 654 - 7384	

Austria	Siemens AG Siemensstraße 88-92 Postfach 83 A-1211 Vienna AUT 1 Hr. Capek Tel. (00431) 2501 - 3779 Fax (00431) 2501 - 3940
Switzerland	Siemens AG Schweiz Freilagerstraße 28-40 Postfach CH-8047 Zürich VHM 1 Hr. Städler Tel. (00411) 495 - 5534 Fax (00411) 495 - 3185
Belgium	Siemens S.A. Charleroiseteenweg 116 B-1060 Brussels VP 4 Mr. Van Overstraeten Tel. (00322) 536 - 2643 Fax (00322) 536 - 2387
Denmark	Siemens A/S Borupvangg 3 DK-2750 Ballerup IP 321 Mr. Saugstrup Tel. (0045) 4477 - 4441 Fax (0045) 4477 - 4016
Finland	Siemens Oy Majurinkatu 6 SF-02601 Espoo TRI TD Mr. Peltola Tel. (003580) 5105 - 3636 Fax (003580) 5105 - 3656
France	Siemens S.A. 39 - 47, Boulevard Ornano F- 93527 Saint Denis Cedex 2 AUT 5 Mr. Weisdorfer Tel. (00331) 4922 - 3913 Fax (00331) 4922 - 3951
Greece	Siemens A.E. Artemidos 8 GR-151 10

Amaroussio/Athen
Department AUT
Mr. Antoniou
Tel. (00301) 6864 - 515
Fax (00301) 6864 - 556

UK

Siemens plc
Princess Road
Manchester, M20 8UR
Energy&Automation
Mr. A. Roworth
Tel. (00461) 446 - 5233
Fax (00461) 446 - 5232

Italy

Siemens S.p.A.
Via Lazzaroni 3
I-20124 Milano
A522
Mr. Vigo
Tel. (00392) 6676 - 2764
Fax (00392) 6676 - 2820

Netherlands

Siemens Nederland N.V.
Prinsesbeatrixlaan 26
NL-2595 Al Den Haag
APS
Mr. Penris
Tel. (003170) 333 - 3515
Fax (003170) 333 - 3496

Norway

Siemens A/S
Ostre Aker vei 90,
Linderud,
Boks 10 Veitwet
N-0518 Oslo
Dept. Industrie-K7
Mr. A. Eggen
Tel. (004722) 63 - 409

Spain

Siemens S.A.
Ronda de Europa 5
E-28760 Tres Cantos
Madrid
AUT 1
Mr. Toledano
Tel. (00341) 803 - 1200
Fax (00341) 803 - 2271

Outside Europe

AUS	Richmond, Victoria Mr. Gough	(0061) 3/420-7218
RSA	Johannesburg Mr. Hillermann	(0027) 11/407-4815
TAI	Taipai Mr. Gulden	(00886) 2/705-4888
USA	Alpharetta, GA	(001) 404/740-3959

Mr. Crew